Design Document for COMPSCI 4ZP6

RatBAT: Rat Behavioral Analysis Tool

Team 8

Brandon Carrasco
Daniel Locke
Jamie Wong
Inoday Yadav

Table of Contents

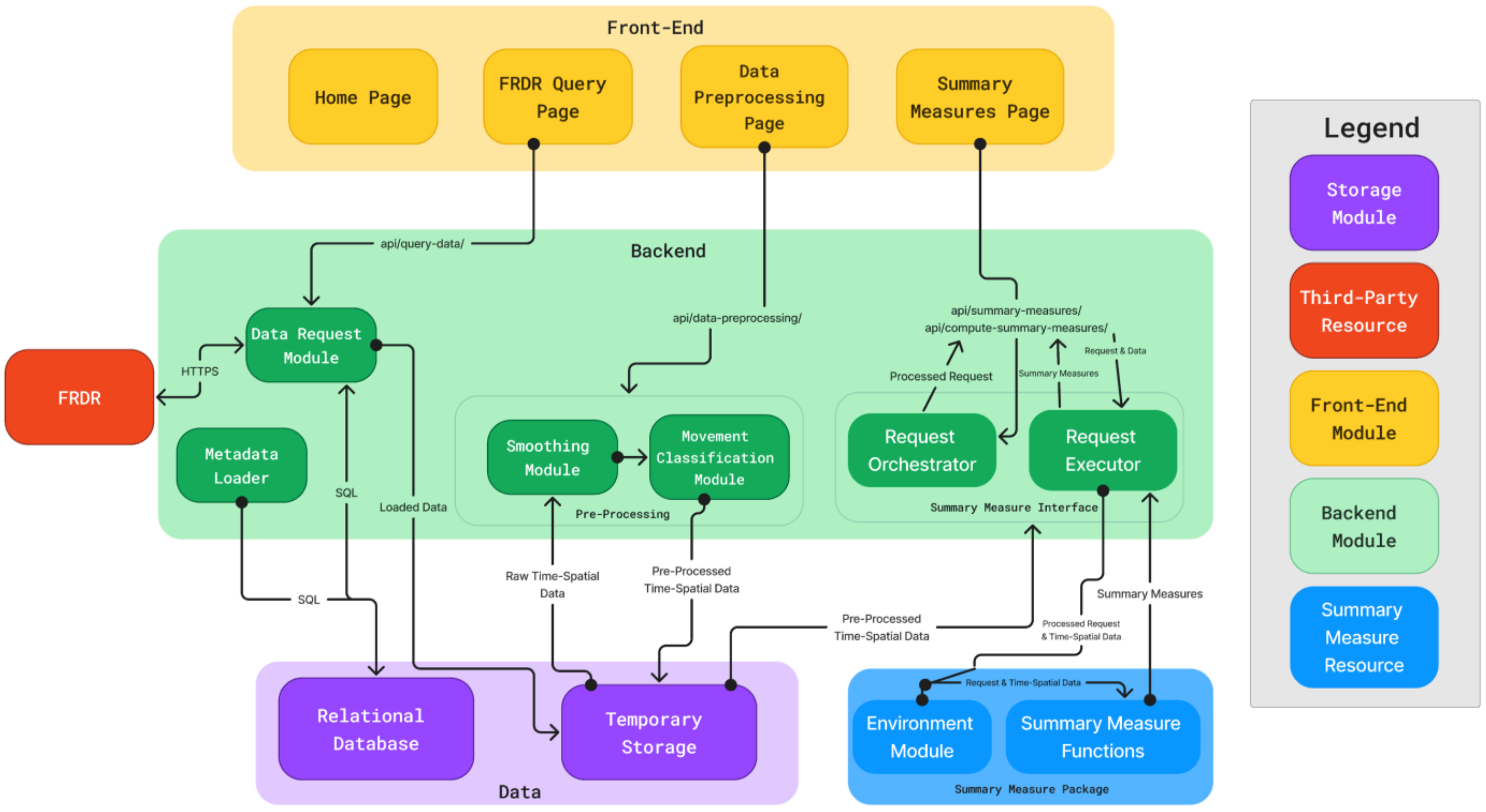
1 Revision History

Date	Version	Notes
1/24/2025	0	Original draft of document
04/04/2025	1	Final version of document

2 Purpose Statement

Our project aims to develop a robust and accessible open-platform web application for researchers. This application will facilitate the preprocessing, computation of summary measures, analysis, and collection of rodent behavioural data from the FRDR repository.

3 Component Diagram



Figma Link

4 Relationship between Components and Requirements

Component	Requirements Covered
Summary Measures Interface	P0: The system must allow users to compute summary measures on temporarily stored pre-processed time series data. P2: The system must allow users to input their own Python-based summary measure algorithms onto the website.
Summary Measures Package	P0: The system must allow users to compute summary measures on temporarily stored pre-processed time series data.
Backend	P0: The system must allow users to compute summary measures on temporarily stored pre-processed time series data. P0: The system must temporarily store downloaded data. P0: The system must temporarily store computed summary measures. P0: The system must temporarily store pre-processed time series data.

	P1: The system must return stored preprocessed data or computed summary measures when the computed data already exists within the database.
UI	P0: The system must allow users to download any temporarily stored data to their local machine. P2: The system must allow users to input their own Python-based summary measure algorithms onto the website. P3: The system must display raw video data on the website when selected by the user to preview. P3: The system must display trajectory plots on the website when selected by the user to preview.
Data Storage	P0: The system must allow users to filter data by project, experiment, trial, or any metadata variables. P0: The system must temporarily store downloaded data. P0: The system must temporarily store computed summary measures. P0: The system must temporarily store pre-processed time series data. P1: The system must store any preprocessed time series data and summary measures computed on the web platform within a database. P1: The system must return stored preprocessed data or computed summary measures when the computed data already exists within the database. P2: The system must store summary measures algorithms created by a user to be accessible for future use via that user's credentials.
FRDR Query	P0: The system must allow users to search for specific projects, experiments, or trials within the library. P0: The system must allow users to filter data by project, experiment, trial, or any metadata variables. P0: The system must allow users to choose which raw data types to download. P0: The system must allow users to download all data that exists in the library. P1:The system must display any raw or pre-processed time series selected by the user.
Data Preprocessing	P0: The system must allow users to apply a standard preprocessing method to time series data. P2: The system must allow users to undo any processing steps performed to data.

5 Component Descriptions

5.1 Summary Measure Interface

Note: For the API implementation of the Summary Measure Interface and Package, please see the Appendix.

Karpov (*Request Orchestrator -* *DependenciesSM.py*)	
Normal Behaviour	This sub-component receives a request from the web platform — request contents created by the user — and ensures it satisfies all dependencies and creates an efficient list of calculations for the Commander sub-component to use. Satisfaction of dependencies would mean all summary measures a part of the initial request has any summary measures they're dependent on added to the calculation list before all dependent summary measures are to be calculated (in the right order).
Implementation	**Karpov** will receive a request for the computation of a ***list of summary measures*** (in the form of a list of strings) from *api/compute-summary-measures*. This request is run through the **ResolveDependencies()** method, which: 1) Adds and re-orders summary measures if needed to satisfy their dependencies 2) Finds and records common calculations between requested summary measures The processed request is passed back to *compute-summary-measures* for calculation of the summary measures. Additionally, summary measure dependencies and common calculations are defined here as constant variables.

Potential Undesired Behaviour	Dependencies not being added or re-ordered properly, resulting in runtime error when passed to the **Commander** instance. Wrong dependencies are defined, leading to runtime error. Common calculations are not located or are incorrectly defined, leading to inefficient calculations.
Commander (*Request Executor -* *CommanderSM.py*)	
Normal Behaviour	This sub-component receives a processed request from **Karpov** and performs all pre-calculations (of common calculations) and calculations of summary measures. These summary measures are calculated using the functions provided by the Summary Measures Package and the final results are returned to the web platform that originally called it.
Implementation	Each **Commander** instance has attributes that store the desired *environment* (of the experiment) and calculated summary measures and transient useful data. Upon receiving a request, the **Commander** instance will calculate all common calculations discovered by **Karpov** and store them as transient useful data in its ***storedAuxiliaryInfo*** attribute (via the **PerformPreCalculations()** method). The **Commander** instance will then proceed through the list of ordered summary measures, calculating them and storing them in its ***calculatedSummaryMeasures*** attribute (via the **CalculateSummaryMeasures()** method). These functions are called generally, which is made possible due to all functions sharing the same call definition scheme. If a function uses calculations that have already been pre-calculated, they will use that data instead of calculating it all over again. Once all summary measures have been calculated, the **Commander** instance will return them to *compute-summary-measures* as an informative dictionary.
Potential Undesired Behaviour	**Commander** instance is instantiated with the wrong environment, outputting incorrect summary measure information.

5.2 Summary Measures Package

Summary Measure Functions (*FunctionalSM.py*)	
Normal Behaviour	The summary measures function package defines the summary measures that a user can request. A **Commander** instance will call functions from this package and return their results to the user.
Implementation	Each function takes in time-series data, an ***Environment*** instance, and any necessary summary measure calculations (as a dictionary), with the optional inclusion of a list/dict of pre-calculations already done beforehand. Should the pre-calculations be present, the functions will use those. Each function will return the summary measure(s) desired.
Potential Undesired Behaviour	Despite the necessary pre-calculations being passed, a summary measure function still re-calculates them as if they don't exist.
Environment Module *(FieldSM.py)*	
Normal Behaviour	The environment module, to be used in conjunction with the Summary Measure functions, has critical information, objects, and methods used for summary measure calculation. It defines the environment itself and the location of specimens inside of it.
Implementation	Contains the ***LOCALE_MAPPING*** variable, matching under-the-hood locale indexing to the Eshkol-Wachmann locales. Also implements the **Environment** class, the test environment for a given experiment, which has the critical **SpecimenLocation()** method. Whenever it's necessary for a summary measure to calculate specimen locale information, this method is called. Additionally, instantiates all the environments used throughout the experiments.
Potential	**SpecimenLocation()** could return the wrong locale in the case of a specimen's location drifting

Undesired Behaviour	over the boundaries between two locales.

5.3 UI / Frontend

Normal Behavior	Users can navigate seamlessly between different pages. The interface provides access to the Home page, FRDR Query, Data Preprocessing, and Summary Measures functionalities. Users can input/select data or parameters, view real-time updates, and retrieve/download results.
Structure	Inputs**:** User actions Outputs**:** Rendered HTML/CSS/JavaScript elements, API responses displayed in UI
Implementation *Subcomponent: ComputeSummary Measures.jsx* Purpose: Computes and displays summary measures results.	Key state variables: ● results: Stores computed summary measure results. ● selectedSummaryMeasures: Tracks which summary measures are selected by the user. ● dataFiles: Holds available data files fetched from the backend. ● selectedDataFile: Indicates the currently selected data file. ● selectedResults: Tracks which results are selected for download. ● summaryMeasuresOptions: Contains all summary measures available for selection. Methods: ● fetchDataFiles and fetchSummaryMeasures: Fetch data files and summary measure options from backend endpoints. ● handleSummaryMeasureToggle: Updates selectedSummaryMeasures by toggling the inclusion of a measure. ● handleApply: Sends selected data file and summary measures to the backend for processing. ● handleDownloadSelected: Converts selected results to CSV format and triggers a download. ● handleMouseDown, handleMouseMove, handleMouseUp: Manages draggable modal functionality. Relationships: ● Data File Selector: Displays fetched files and lets users select one. ● Summary Measure Selector: Displays fetched measures and lets users select multiple options. ● Result Section: Displays and allows interaction with computed results.
Implementation Subcomponent: *Home.jsx* Purpose: Provides users with a brief overview of the research project and how to navigate the web platform	Key State Variables: There are no state variables in this component because it primarily focuses on static content and navigation. Methods: ● Static Navigation Buttons: The onClick handlers in the Call to Action Section (cta-section) navigate to specific routes: ○ /frdr-query: Directs to the FRDR query page for selecting data. ○ /data-preprocessing: Directs to the data preprocessing page. ○ /compute-summary-measures: Directs to the summary measures page
Implementation Subcomponent: *DataPreprocessing .jsx*	Key State Variables: ● selectedMethods (Array): Tracks which preprocessing methods are selected by the user. Initialized as an empty array.

Purpose: Provides an interface for users to preprocess data before running summary measures.	• selectedDataFile (String): Stores the name of the currently selected data file. Initialized as an empty string. • selectedResults (Array): Tracks which result items are selected by the user. Initialized as an empty array. Methods: • handleMethodToggle(method): Toggles the selection of a preprocessing method. Adds the method to selectedMethods if not already selected; removes it otherwise. • handleDataFileChange(event): Sets the selected data file based on user input. • handleResultToggle(result): Toggles the selection of a result item. Adds the result to selectedResults if not already selected; removes it otherwise. • handleSelectAllResults(): Selects all result items by setting selectedResults to the complete results array. • handleApply(): Validates that a data file is selected before proceeding. Alerts the user that the "Apply" button has been clicked. • handleDownloadSelected(): Alerts the user that the "Download Selected" button has been clicked. Relationships: • Data File Selection & Apply: The user must select a data file (selectedDataFile) before applying preprocessing methods. This relationship enforces logical order in user interaction. • Preprocessing Methods & Results: Preprocessing methods (selectedMethods) impact the potential results displayed to the user. • Result Selection & Download: The results selected (selectedResults) determine what will be downloaded when the "Download Selected" button is clicked.
Implementation *Subcomponent: FRDRQuery.jsx* Purpose: Enables users to query and fetch data from the FRDR (Federal Research Data Repository).	Key state variables: • data: Stores the filtered dataset received from the backend. • filters: Stores the filter criteria selected by the user, and sent to the backend for querying. • searchTerm: Stores the user's input for filtering • toggledButtons: Tracks the toggle state for each meta filter. Methods: • results: Stores computed summary measure results. • handleToggle(index): Toggles the state of a specific meta button. Updates the toggledButtons state. • handleSearchChange(event): Updates the searchTerm state when the user types in the search bar. • handleDropdownChange(filterKey, value): Updates the filters from a selected discrete filter category. • handleInputChange(filterKey, value): Updates filters state depending on user input for typed filters. • handleDownload(): Sends a request to the backend to generate a downloadable JSON file. • applyFilters(newFilters): Updates the filter state with new filter criteria. • fetchFilteredData(searchTerm, filters): Sends API request to backend with searchTerm and filters as query parameters. Relationships: • Filter Window: Displays filter parameter and passes the user's input back through applyFilter to adjust the filters state. • Data Window: Displays the filtered dataset (data) in a scrollable window
Potential undesired behaviours	• Slow or large results rendering causing UI lag. • API failures: Backend endpoints are unreachable or return errors. • UI bugs: Modal is unresponsive or misaligned after dragging.

5.4 Data Storage

Database Structure (*db_connector/models.py)*

Normal Behaviour	Database consisting of 21 tables interconnected by foreign key relationships. 20 of these tables store the relationships between the many independent variables in the library while the last table is dedicated to storage of the time series data.
Implementation	The database is created in MariaDB and managed via the Django backend. Tables are defined as **Django model classes** (django.db.models.Model) which are then automatically migrated to the database by Django. All database models are accessible via **db_connector.models**.
API	All database interaction is done via backend using **db_connector.models** which can be used to create queries that are automatically converted into SQL by Django and used to interface with the database.
Potential Undesired Behaviour	Some data in the library may not match the expected type leading to issues when trying to load data into the database. Unexpected missing fields in the data library could result in primary/foreign key errors if the missing data relates to important fields.
Metadata Loader Command *(load_metadata.py)*	
Normal Behaviour	Initialization command to be executed when setting up the database in order to extract the metadata from its current storage in .csv files, structure it into tables to match the structure of the database and load it into the database.
Implementation	The metadata loader is implemented as a Django management command. The command extracts the data from csv files stored in the project and uses the **pandas** library to structure and modify the data into 20 different tables. The command then utilizes the custom **build_model** function found in **db_helper.py** which takes a Django model class and pandas DataFrame and converts the DataFrame into instances of the model class which are then loaded into the database via Django's **bulk_create** function. If there is already data in the metadata tables, it is deleted to avoid duplicate data.
API	The command is executed via the command line using: '**python backend/manage.py load_metadata'**.
Potential Undesired Behaviour	Certain data cleaning and processing steps could lead to undesired behaviour if the data in certain fields is not all stored in the expected format.

5.5 Backend

Subcomponent*:* *summary_measures/views.py*	
Normal Behaviour	Receives POST requests with data_file_path, summary_measures, and environment parameters. Loads the specified data file, computes summary measures based on the environment configuration, and returns the calculated results.
API and Structure	Inputs: ● api/compute-summary-measures/ (POST): Accepts JSON payload with: ○ data_file_path (string) ○ summary_measures (list of strings) ○ environment (object) ● api/data-files/ (GET): Retrieves a list of available data files. ● api/summary-measures/ (GET): Retrieves a list of available summary measures. Outputs: ● api/compute-summary-measures/ (POST): ○ On success: JSON containing calculated summary measures. ○ On failure: JSON error message with status code. ● api/data-files/ (GET):

	○ On success: JSON list of file names. ○ On failure: JSON error message with status code. ● api/summary-measures/ (GET): ○ On success: JSON list of summary measures. ○ On failure: JSON error message with status code.
Implementation	Class: ComputeSummaryMeasuresView Methods: ● post(self, request, *args, **kwargs): Handles POST requests to compute summary measures. ● Retrieves data_file_path, summary_measures, and environment from request data. ● Loads the specified data file using pandas. ● Uses CommanderSM to calculate summary measures based on the loaded data and environment. ● Returns the calculated summary measures in the HTTP response.
Potential Undesired Behaviour	● Slow computation times ● Errors due to missing or incorrectly formatted input parameters. ● Failure to access or load the data file from the specified path.
Subcomponent: *frdr_query/views.py*	
Normal Behaviour	Receives POST requests with filters, data caching file path, requested data types, and save parameter. Fetches files in the scope of the filter from the local database and FRDR (with the FRDR being used only to access files that don't already exist locally) then saves them at the provided cache location.
API and Structure	**Input -** api/query-data/ (POST): Accepts JSON payload with: ● **filters -** list of strings each containing *field* to filter on, *lookup* type (exact match, contains, less than…), and *value* to use for comparison. ● **cache_path** - file path to location where raw datafiles (videos, pathplots, trackfiles) will be temporarily cached. ● **dtypes** - string specifying which data types should be downloaded from the FRDR ('v'-video, 't'-trackfiles, 'p'-pathplots, 'a'-all, or any combination of v, t, and p) ● **save** - boolean value specifying if fetched time series data is to be stored in the database for long term storage purposes. **Output -** api/query-data/ (POST) ● Success/error message stating the outcome of the load.
Implementation	Class: QueryDataView Methods: ● post(self, request, *args, **kwargs): Handles POST requests to query and retrieve data from the FRDR and local database. ○ Receives POST requests with **filters**, data caching file path (**cache_path**), requested data types (**dtypes**), and boolean save parameter (**save**). ○ Calls on **get_frdr_urls()** to determine what data already exists and what needs to be accessed from the FRDR, calls on **frdr_request()** to fetch needed and save needed data from the FRDR, and loads any requested data that already exists in the database to **cache_path**.
Potential Undesired Behaviour	● Errors in the query/download process that are not properly communicated via the output leading to false success messages.

5.6 FRDR Query

Query functions: FRDRQuery/query.py	
Normal Behaviour	**get_frdr_urls():** Takes **filters**, django model class for the Trial table (**trial_model**), and desired data types (**dtypes**). Returns FRDR urls of any datafiles that are covered under the filters but do not already exist in the local database. **frdr_request():** Takes list of **files**, temp save path (**cache_path**), django **model** class for the time series table, and boolean **save** parameter. Fetches all needed files from the FRDR and saves fetched time series data to the database if requested.
API	**Accessed via** frdr_query/views.py (5.5) **get_frdr_urls() -** Takes 3 parameters ● **filters:** list of strings used to filter database as defined in (5.5) ● **trial_model:** django model class for the model representing the **Trial** database table. ● **dtypes:** string specifying requested data types as defined in (5.5) *Returns:* List of tuples representing files from the FRDR (Trial_ID *(int)*, dtype *(str)*, url *(str)*) **frdr_request() -** Takes 4 parameters ● **files:** list of tuples representing FRDR files as returned by **get_frdr_urls()** ● **cache_path:** file path to temporary save location as defined in (5.5) ● **model:** django model class for the model representing the **TimeSeries** database table. ● **save:** boolean parameter specifying whether to save time series data in the database as defined in (5.5)

Implementation	**get_frdr_urls():**
	Filtering is done using **django.db.models.Q** objects, filters are applied to **trial_model** and returned data is queried from fields **Trial.Trial_ID**, and **Trial.Video**/**Trial.Trackfile**/**Trial.Pathplot** (FRDR urls).
	frdr_request():
	Data is retrieved from the FRDR via HTTP GET requests performed through the python **requests** library. Raw files are then saved at **cache_path** for future use, and if **save** is *true*, time series data is loaded into **pandas** DataFrames, and stored in the database via the custom **build_model** helper function.

Potential Undesired Behaviour	Uncaught errors arising from accessing files from the FRDR or from accessing the local database. Incorrectly applying filters resulting in unnecessary file downloads.

6 User Interface

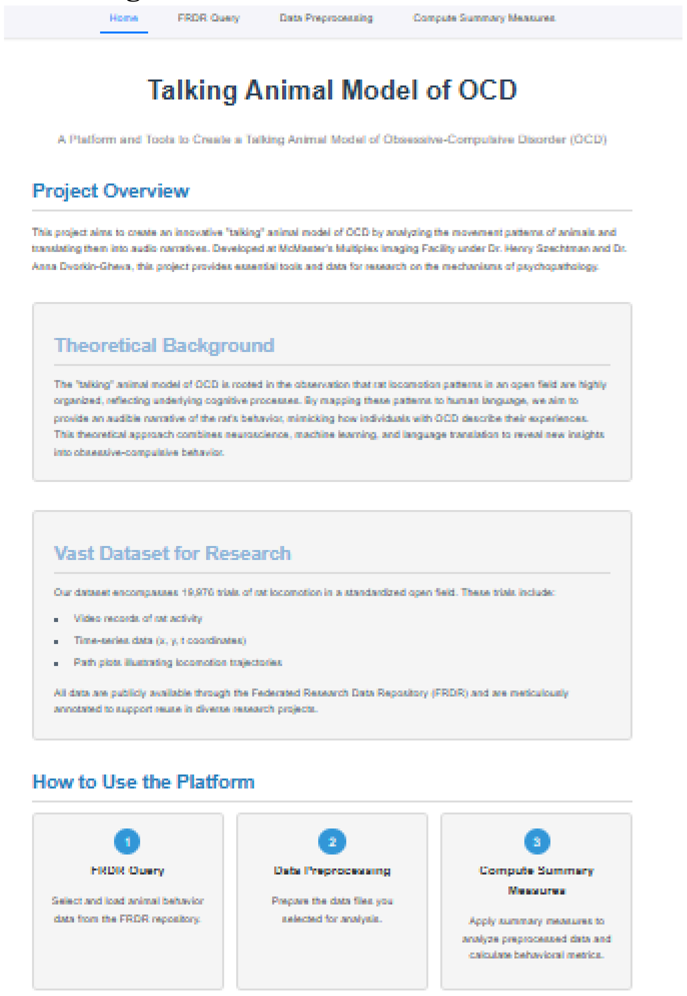
Our web application consists of four web pages: Home, FRDR Query, Data Preprocessing, and Compute Summary Measures. Designed with simplicity in mind for a research-focused environment, our interface features a clean layout and intuitive navigation. Our color palette is a mix of black, grey, white, and blue. Blue is used to highlight some buttons, headings, and the header tab the user is currently on. All regular text is black, some text backgrounds are light gray to indicate them as subheading text in the home page. All pages will also have draggable popup tips to help the user understand how to use the functionalities. Please see the figma for our paper prototype and current implementations in the appendix for more details.

Figma: Link

7 Appendix

7.1 UI Design

Home Page



Talking Animal Model of OCD

A Platform and Tools to Create a Talking Animal Model of Obsessive-Compulsive Disorder (OCD)

Project Overview

This project aims to create an innovative "talking" animal model of OCD by analyzing the movement patterns of animals and translating them into audio narratives. Developed at McMaster's Multiplex Imaging Facility under Dr. Henry Szechtman and Dr. Anna Dvorkin-Gheva, this project provides essential tools and data for research on the mechanisms of psychopathology.

Theoretical Background

The "talking" animal model of OCD is rooted in the observation that rat locomotion patterns in an open field are highly organized, reflecting underlying cognitive processes. By mapping these patterns to human language, we aim to provide an audible narrative of the rat's behavior, mimicking how individuals with OCD describe their experiences. This theoretical approach combines neuroscience, machine learning, and language translation to reveal new insights into obsessive-compulsive behavior.

Vast Dataset for Research

Our dataset encompasses 19,976 trials of rat locomotion in a standardized open field. These trials include:

- Video records of rat activity
- Time-series data (x, y, t coordinates)
- Path plots illustrating locomotion trajectories

All data are publicly available through the Federated Research Data Repository (FRDR) and are meticulously annotated to support reuse in diverse research projects.

How to Use the Platform

1 FRDR Query

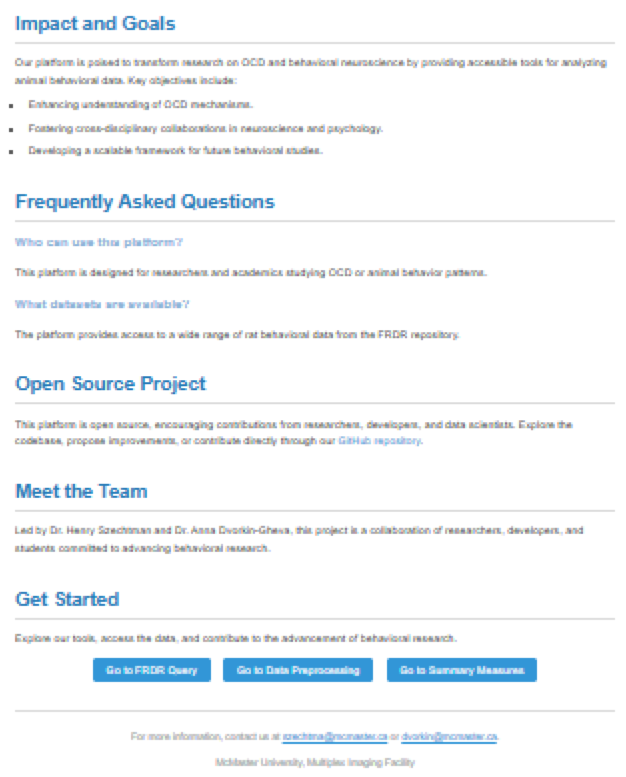
Select and load animal behavior data from the FRDR repository.

2 Data Preprocessing

Prepare the data files you selected for analysis.

3 Compute Summary Measures

Apply summary measures to analyze preprocessed data and calculate behavioral metrics.



Impact and Goals

Our platform is poised to transform research on OCD and behavioral neuroscience by providing accessible tools for analyzing animal behavioral data. Key objectives include:

- Enhancing understanding of OCD mechanisms.
- Fostering cross-disciplinary collaborations in neuroscience and psychology.
- Developing a scalable framework for future behavioral studies.

Frequently Asked Questions

Who can use this platform?

This platform is designed for researchers and academics studying OCD or animal behavior patterns.

What datasets are available?

The platform provides access to a wide range of rat behavioral data from the FRDR repository.

Open Source Project

This platform is open source, encouraging contributions from researchers, developers, and data scientists. Explore the codebase, propose improvements, or contribute directly through our GitHub repository.

Meet the Team

Led by Dr. Henry Szechtman and Dr. Anna Dvorkin-Gheva, this project is a collaboration of researchers, developers, and students committed to advancing behavioral research.

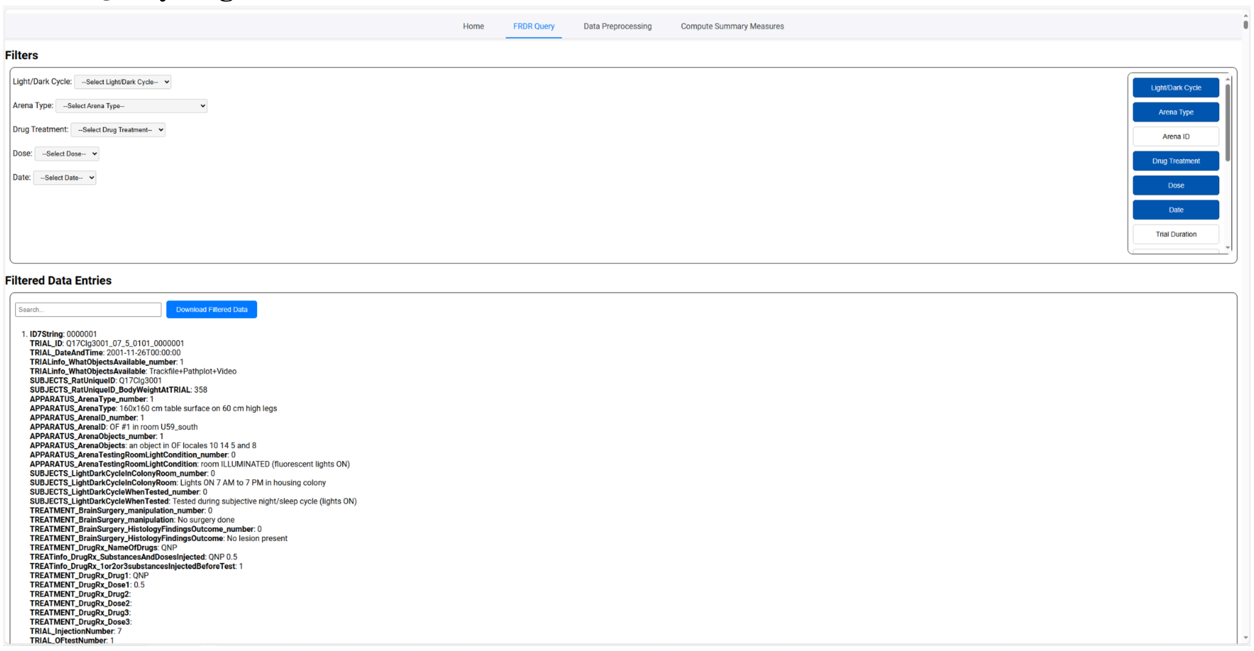
Get Started

Explore our tools, access the data, and contribute to the advancement of behavioral research.

For more information, contact us at szechtma@mcmaster.ca or dvorkin@mcmaster.ca.

McMaster University, Multiplex Imaging Facility

FRDRQuery Page



Filters

Light/Dark Cycle: --Select Light/Dark Cycle--

Arena Type: --Select Arena Type--

Drug Treatment: --Select Drug Treatment--

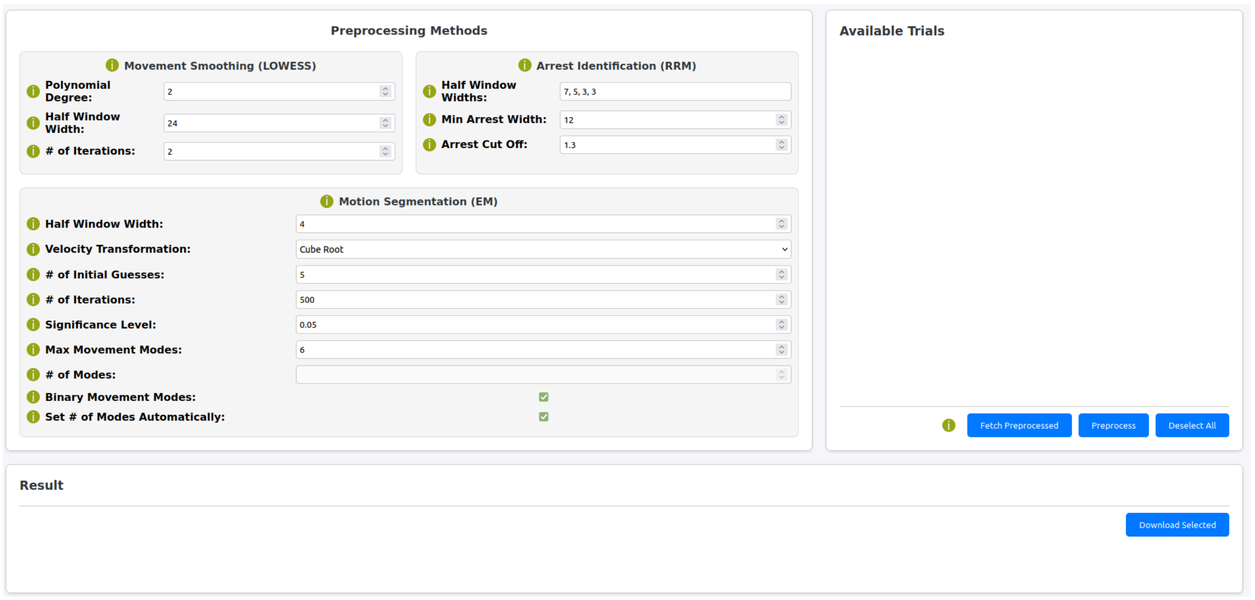
Dose: --Select Dose--

Date: --Select Date--

Filtered Data Entries

1. **ID7String**: 0000001
   TRIAL_ID: Q17Clg3001_07_5_0101_0000001
   TRIAL_DateAndTime: 2001-11-26T00:00:00
   TRIALinfo_WhatObjectsAvailable_number: 1
   TRIALinfo_WhatObjectsAvailable: Trackfile+Pathplot+Video
   SUBJECTS_RatUniqueID: Q17Clg3001
   SUBJECTS_RatUniqueID_BodyWeightAtTRIAL: 358
   APPARATUS_ArenaType_number: 1
   APPARATUS_ArenaType: 160x160 cm table surface on 60 cm high legs
   APPARATUS_ArenaID_number: 1
   APPARATUS_ArenaID: OF #1 in room U59_south
   APPARATUS_ArenaObjects_number: 1
   APPARATUS_ArenaObjects: an object in OF locales 10 14 5 and 8
   APPARATUS_ArenaTestingRoomLightCondition_number: 0
   APPARATUS_ArenaTestingRoomLightCondition: room ILLUMINATED (fluorescent lights ON)
   SUBJECTS_LightDarkCycleInColonyRoom_number: 0
   SUBJECTS_LightDarkCycleInColonyRoom: Lights ON 7 AM to 7 PM in housing colony
   SUBJECTS_LightDarkCycleWhenTested_number: 0
   SUBJECTS_LightDarkCycleWhenTested: Tested during subjective night/sleep cycle (lights ON)
   TREATMENT_BrainSurgery_manipulation_number: 0
   TREATMENT_BrainSurgery_manipulation: No surgery done
   TREATMENT_BrainSurgery_HistologyFindingsOutcome_number: 0
   TREATMENT_BrainSurgery_HistologyFindingsOutcome: No lesion present
   TREATMENT_DrugRx_NameOfDrugs: QNP
   TREATinfo_DrugRx_SubstancesAndDosesInjected: QNP 0.5
   TREATinfo_DrugRx_1or2or3substancesInjectedBeforeTest: 1
   TREATMENT_DrugRx_Drug1: QNP
   TREATMENT_DrugRx_Dose1: 0.5
   TREATMENT_DrugRx_Drug2:
   TREATMENT_DrugRx_Dose2:
   TREATMENT_DrugRx_Drug3:
   TREATMENT_DrugRx_Dose3:
   TRIAL_InjectionNumber: 7
   TRIAL_OFtestNumber: 1

Data Preprocessing Page



Preprocessing Methods

Movement Smoothing (LOWESS)

Polynomial Degree:	2
Half Window Width:	24
# of Iterations:	2

Arrest Identification (RRM)

Half Window Widths:	7, 5, 3, 3
Min Arrest Width:	12
Arrest Cut Off:	1.3

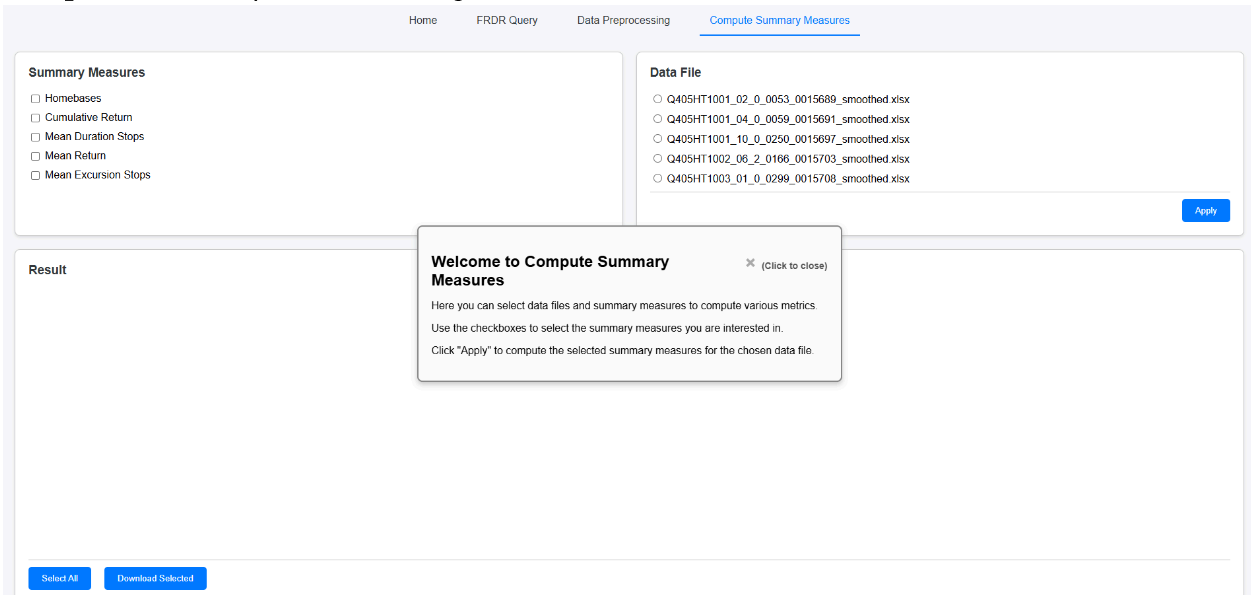
Motion Segmentation (EM)

Half Window Width:	4
Velocity Transformation:	Cube Root
# of Initial Guesses:	5
# of Iterations:	500
Significance Level:	0.05
Max Movement Modes:	6
# of Modes:	
Binary Movement Modes:	✓
Set # of Modes Automatically:	✓

Available Trials

Result

Compute Summary Measures Page



Summary Measures

- ☐ Homebases
- ☐ Cumulative Return
- ☐ Mean Duration Stops
- ☐ Mean Return
- ☐ Mean Excursion Stops

Data File

- ○ Q405HT1001_02_0_0053_0015689_smoothed.xlsx
- ○ Q405HT1001_04_0_0059_0015691_smoothed.xlsx
- ○ Q405HT1001_10_0_0250_0015697_smoothed.xlsx
- ○ Q405HT1002_06_2_0166_0015703_smoothed.xlsx
- ○ Q405HT1003_01_0_0299_0015708_smoothed.xlsx

Result

Welcome to Compute Summary Measures

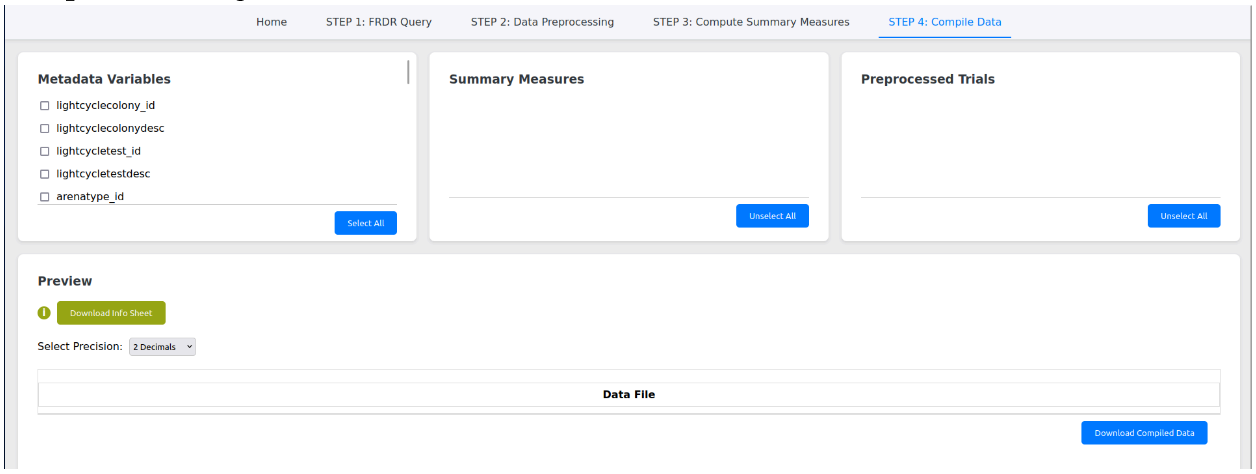
(Click to close)

Here you can select data files and summary measures to compute various metrics.

Use the checkboxes to select the summary measures you are interested in.

Click "Apply" to compute the selected summary measures for the chosen data file.

Compile Data Page



Metadata Variables

- ☐ lightcyclecolony_id
- ☐ lightcyclecolonydesc
- ☐ lightcycletest_id
- ☐ lightcycletestdesc
- ☐ arenatype_id

Summary Measures

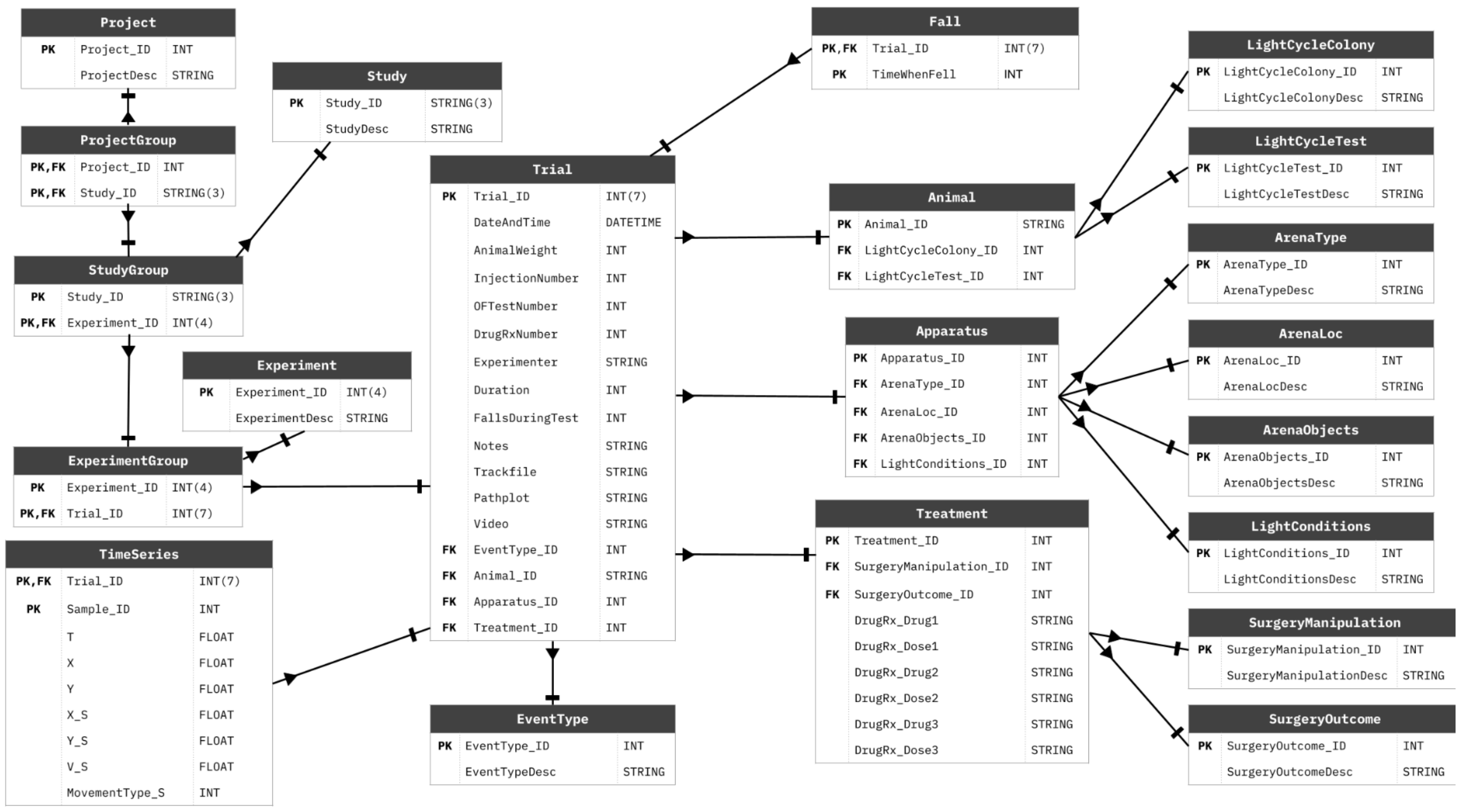
Preprocessed Trials

Preview

Select Precision: 2 Decimals

Data File

7.2 Database Design



Link to diagram (PDF)

7.3 Implementation Details

7.3.1 Summary Measures

DependenciesSM.py	
SM_DEPENDENCIES	Dict of summary measure reference ids (*Strings*) as keys. Its values are a list of strings (of other SM reference ids). Specifies the prerequisite SMs that must be calculated for a given SM to be calculated.
DATA_DEPENDENCIES	Dict of SM ref. Ids (*Strings*) as keys. Its values are a list of *strings* (of ref. Ids for common calculation functions). Specifies which common calculations an SM performs. A common calculation is a calculation that appears in other SMs as well. Used for pre-calculations by the **Commander** class.

Karpov	
+ ResolveDependencies()	Adds prerequisite SMs & re-orders the list of requested SMs such that all prerequisite SMs are calculated before their dependent SMs. Additionally, creates a list of common calculations that should be performed based on the final list of SMs. <u>Parameters</u> - summary_measures - List of strings (of reference ids of SMs) <u>Returns</u> - List of strings (Re-ordered & proofed SM request) - List of strings (desired common calculations)
- AddRequiredSummaryMeasures()	Given a list of SM ref. Ids, adds any prerequisite SM ref. Ids to the list of summary measures to be calculated. <u>Parameters</u> - summary_measures - List of strings (of reference ids of SMs) <u>Returns</u> - List of strings (of reference ids of SMs)
- OrderSummaryMeasures()	Given a list of SM ref ids, re-order them such that all pre-req SMs come before their dependent SMs. <u>Parameters</u> - summary_measures - List of strings (of reference ids of SMs) <u>Returns</u> - List of strings (of reference ids of SMs)

FunctionalSM.py	
Defines all summary measure and common calculations functions to be used by the **Commander** class.	
Summary Measure Functions	Utilizes a common input schema: 1. data : Matrix of Time-Spatial data 2. env: Environment instance 3. pre_reqs: Dict (ref. Ids. -> Calc'd SMs) 4. pre_calcs: Dict (ref. Ids -> data) Returns one or more values.
Common Calculation Functions	Takes data & env (specified above). Returns auxiliary data (integer, float, list, etc.).

CommanderSM.py	
SM_MAPPING ***DATA_MAPPING***	Maps reference ids (for SMs & common calcs; *Strings*) to function names (to calculate SMs and common data; Strings).
Commander	
- *env* Experiment's **Environment** - *storedAuxiliaryInfo* Pre-calculated data	- *calculatedSummaryMeasures* Calculated SMs
- SelectEnvironment()	Sets the environment based on the name passed to it on instantiation. Parameters - environmentName - String (reference id of the desired environment). Returns - None
- PerformPreCalculations()	Performs & records all common calculations in the *storedAuxiliaryInfo* variable. Parameters - common_calculations - List of Strings (reference ids of desired common calculations) Returns - None
+ CalculateSummaryMeasures()	Given data, SMs, & common calcs, calculates & returns dict of request summary measures. Performs pre-calculations first before calculating summary measures. Parameters - data - Matrix of Floats (time-spatial data; see the bottom for what each column corresponds to). - summary_measures - List of Strings (reference ids of desired summary measures) - common_calcs - List of Strings (reference ids of desired common calculations)

	Returns - Dictionary (mapping SM ref. Ids to ~ data-col-1 = Frame data-col-2 = x-coordinate data-col-3 = y-coordinate data-col-4 = velocity data-col-5 = segment type (lingering vs. progression)

FieldSM.py	
LOCALE_MAPPING	Maps list indices (generated by **Environment's** SpecimenLocation method) to the Eshkol-Wachmann locales (*Integers*).
COMMON_ENV	**Environment** instance defining the test environment that most experiments used (**Environment**).
+ GetLocaleFromIndex()	Given an index, return the locale that the index maps to in ***LOCALE_MAPPING***. Parameters - index - Integer (between 0 to 24). Returns - Integer (the Eshkol-Wachmann locale that it the index corresponds to)
PhysicalObject	
A physical object that exists within an environment. Currently implemented by **Rectangle** & **Polygon** subclasses.	
- points List of x- & y-coordinates of the object's vertices.	
+ is_within()	Checks if the specimen is inside of an object given its x & y coordinates. Parameters - x - Float (x-coordinate of the specimen). - y - Float (y-coordinate of the specimen) Returns - Boolean (if the specimen is inside of

	the object)
Environment	
- *grid* NumPy meshgrid implementing the locale grid. X & y coordinates are between 20 & 180, meaning the grid is a 160 x 160 meshgrid. - *objects* List of **PhysicalObject** instances.	
- GenerateGrid()	Given a list of vertical and horizontal line coordinates, return a meshgrid to simulate the environment's locale grid. Parameters - lineCoords - List(s) of tuples (tuples specifying start and end points for horizontal and vertical lines making up the grid). Returns - Meshgrid (grid making up the environment of the experiment.)
+ SpecimenLocation()	Given x & y coordinates of the specimen, either return the locale mapping or index (that could map to a locale) it's located in. Parameters - x - Float (x-coordinate of the specimen). - y - Float (y-coordinate of the specimen) - index - Boolean (True if meshgrid index is desired; False if Eshkol-Wachmann local is desired). Returns - Integer (the Eshkol-Wachmann locale or index of the meshgrid that the specimen is located in)